LLNL-CONF-608492

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Probability of Initiation and Extinction in the Mercury Monte Carlo Code

M. S. McKinley, P. S. Brantley

December 17, 2012

International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering
Sun Valley, ID, United States
May 5, 2013 through May 9, 2013

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering (M&C 2013), Sun Valley, Idaho, USA, May 5-9, 2013, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2013)

PROBABILITY OF INITIATION AND EXTINCTION IN THE MERCURY MONTE CARLO CODE

M. S. McKinley and Patrick S. Brantley
Lawrence Livermore National Laboratory
7000 East Ave., Livermore, CA 94551
mckinley9@llnl.gov; brantley1@llnl.gov

ABSTRACT

A Monte Carlo method for computing the probability of initiation has previously been implemented in Mercury. Recently, a new method based on the probability of extinction has been implemented as well. The methods have similarities from counting progeny to cycling in time, but they also have differences such as population control and statistical uncertainty reporting. The two methods agree very well for several test problems. Since each method has advantages and disadvantages, we currently recommend that both methods are used to compute the probability of criticality.

Key Words: Monte Carlo, probability, initiation, extinction, Mercury

1. INTRODUCTION

In a supercritical nuclear system, a single neutron could lead to a divergent chain reaction. Alternatively, the single neutron could lead to a finite number of neutrons which in turn go extinct through absorption and leakage. The probability of initiation (*poi*) is the probability that one neutron from a specified source leads to a divergent chain reaction, while the probability of extinction (*poe*) is the probability that it does not ($1 - poi$).

Previous research in this field involved a proper modeling of the spatially dependent poi using adjoint equations from a deterministic, discretized solution.[1] Monte Carlo solutions for poi where introduced which introduced a “cutoff” population that represented divergence of solution.[2] Mercury [3], a Monte Carlo particle transport code developed at Lawrence Livermore National Laboratory, has a *poi* capability based on a forward solution to the Boltzmann transport equation [4].

The *poi* method in Mercury begins with a source of particles, which represent the heads of unique families. Each time cycle, the number of progeny for each family is calculated. Once the number of progeny has surpassed a predefined user-supplied threshold, the family is tallied as divergent and the whole family is removed. The probability of initiation is the number of divergent families divided by the number of sourced families.

Recently, Booth has suggested the use of a *poe* estimate using Monte Carlo [5] and implemented it in a modified version of MCNP [6,7]. One of the main advantages of the *poe* method over the *poi* method is that it does not require a user to specify the number of progeny required to determine if a chain is divergent.

A *poe* based solution for determining the probability of initiation has recently been implemented in Mercury. This paper describes the implementation of the *poi* and *poe* algorithms in mercury as well as extensions to the *poe* algorithm to accommodate spatial parallelism via domain decomposition. The results of the *poi* and *poe* algorithms are compared for both an analytic test problem and a suite of test problems. We compare the computational efficiency of the two methods for a uranium sphere test problem. We conclude with a discussion and suggestions for future work.

2. DEVELOPMENT

2.1. Time Versus Generation Cycling

In Mercury, the *poi* calculation focuses on the estimation of a divergent chain of neutrons by counting the number of progeny from a single sourced neutron. Since Mercury has spatial parallelism via domain decomposition [8], one source neutron may have progeny that spread to other domains as the particle moves. Since the information on family history may spread across many domains, a truly static method cannot be employed without using an abundant amount of parallel communications that would increase the run time. To get around this hurdle, the size of the particle family is now determined at certain stopping conditions that apply for all processes running. Two obvious choices for collecting family sizes are stopping at certain simulation times or stopping at certain number of generations. Generation cycling has previously been looked at in MCNP with the probability of initiation algorithm [7].

While the *poi* calculation stops at certain simulation times (up to a maximum time) to collect family sizes, an attempt was made to compute the *poe* calculation by stopping every generation of the sourced neutrons. A generation of neutrons ends once all have leaked, been absorbed or produced offspring in a reaction. The idea behind using this approach for *poe* was that every generation would contribute real change since every particle is lost or reproduces. In addition, some particles could potentially take long leaps in time and may give a faster answer than time stepping. Finally, simulation particle growth would be limited by only a factor of the effective *k* eigenvalue. Unfortunately, testing showed that generation cycling performed worse in terms of speed and noise than time cycling for many cases and did not offer a big enough advantage to warrant keeping it in the code.

2.2. Population Control

Since the problem is supercritical, the number of particles in the problem will increase over time, which leads to a need to reduce the number of simulation particles over time. The *poi* calculation has a natural population control which removes chains of families once they reach their progeny threshold. Unfortunately, if a large threshold is used, the population could grow quite large and overflow memory before families are purged. Lowering this threshold could save memory, but may result in worse accuracy. A further *poi* population control option exists, but it appears to be sub-optimal in its performance.

The *poe* mode has no natural population control, but Booth [5] demonstrates useful variance reduction techniques. In Mercury, each family is assigned an importance based on p^C, where p is the estimated probability of extinction (with a minimum value of 0.5 to avoid excessive changes in weight) and C is the number of progeny. This importance roughly represents the probability that the family will go extinct. Families split or undergo Russian roulette with these assigned importances to match the target number of simulation particles. The overall impact is that families with a large number of progeny will have low importance and thus high weight. In addition, most of the simulation particles will be focused on smaller families.

An additional feature implemented for *poe* is a safety option, which is on by default. This safety option limits splitting to at most a ratio of 2 to 1 split and Russian roulette to a 50% survival probability. This option seeks to limit some of the huge changes that may reduce the effectiveness of a variance reduction scheme.

2.3. Counting Progeny

Due to spatial domain decomposition, counting the progeny per family is a challenging task. The *poi* and *poe* implementations methods have different solutions to this issue. The *poi* method creates an array over all families, which is the number of sourced neutrons. A parallel reduction is performed to sum all the members of each family on every processor. The disadvantage to this implementation is that variance reduction would be more difficult to fully implement due to the need to avoid gaps in family identifiers.

The *poe* method was implemented with variance reduction in mind. At census, every particle record is placed in an array of structures. Each structure stores the family identifier number, weight of the family, and number of members of the family that have been found. This array undergoes local and then global sorting based on family identifier number. When family members are located next to each other in the array, their data structure merges into a single structure with an increased family member count. The end product is an array over all families that are still alive. This list is used to determine family importances so all processors are synchronized for splitting and Russian rouletting.

2.4. Probability Estimation

On the surface, it would seem simple to calculate the *poe* as in Eq. (1):

$$poe = 1 - \frac{W_s}{W_0} \tag{1}$$

where W_s is the surviving family weight and W_0 is the initial sourced in weight. However, this simple approach turns out to be less effective once variance introduction is involved. Russian roulette is more likely to happen on families with a large number of progeny, so their weights can grow quite large and represent a statistical uncertainty due to the random nature of Russian roulette.

To solve this issue, the number of families that go extinct is instead tallied. Since the weights tend to be small for families that go extinct, the extinction weight has a lower statistical uncertainty. Calculating the number of families that go extinct is nontrivial due to spatial domain decomposition. So the array of surviving families is kept from the previous cycle, and missing family identifiers are tallied as extinct families.

Since the *poi* and *poe* implementations in Mercury both use time cycling, an uncertainty always exists in how to treat the surviving families that have not reached the critical threshold to mark them as divergent. Both methods treat these families as failures when computing their probability. However, the *poe* method estimates a projected probability by assuming that each family could go extinct with a probability of poe^C, where C is the number of progeny in that family.

2.5. Batch Statistics

One of the features implemented for *poe* is batch statistics. The user specifies how many batches will be used, N. Each batch is assigned W_0/N starting particle weight and computes its own *poe*. These batch *poe* calculations are used to compute the standard deviation and average probability. This feature has not yet been implemented in the *poi* calculation.

3. TEST RESULTS

3.1. Analog Test

Booth [5] introduced a simple analytical infinite medium problem that only has two reactions. The probability per collision for an absorption is 49.9% while the probability for fission is 50.1%. Fissions always produce 2 neutrons. The analytic *poi* for this test problem is 3.992016×10^{-3}.

Table I. Analytic test results

N[a]	poi	Fractional Error	poe's poi[b]	Fractional Error
2^{10}	3.906E-03	-0.021	2.138E-03	-0.464
2^{11}	5.371E-03	0.345	4.147E-03	0.039
2^{12}	3.906E-03	-0.021	2.893E-03	-0.275
2^{13}	3.662E-03	-0.083	3.710E-03	-0.071
2^{14}	5.005E-03	0.254	4.782E-03	0.198
2^{15}	4.456E-03	0.116	4.640E-03	0.162
2^{16}	4.211E-03	0.055	4.478E-03	0.122

2^{17}	4.120E-03	0.032	4.268E-03	0.069
2^{18}	3.944E-03	-0.012	4.056E-03	0.016
2^{19}	3.937E-03	-0.014	4.046E-03	0.014
2^{20}	3.966E-03	-0.006	4.029E-03	0.009
2^{21}	3.965E-03	-0.007	3.977E-03	-0.004
2^{22}	---	---	4.016E-03	0.006

[a]Number of initial families.
[b]The *poi* is calculated as 1-*poe*.

The Mercury results for *poi* and *poe* are shown in Table 1. Both methods show convergence towards the analytic solution with increasing number of starting families. This trend loosely follows Booth's results [5]. Differences with Booth's results may be due to the time cycling in Mercury's implementation compared to the history-based approach in Booth's implementation. There are no results for *poi* over 2^{21} families due to running out of computer memory and for *poe* over 2^{22} due to computer run time. These are not hard limits as the calculations could have been run on more processors or with more memory to go further, but the trend is pretty evident.

3.2. Uranium / Plutonium Sphere Test

A suite of test problems developed by LANL and LLNL in the 1970's previously served as a test basis for the probability of initiation implementation in Mercury[4]. The suite contains spheres of uranium and plutonium of varying radius and density. The suite was rerun for the *poi* method to compute a statistical uncertainty based on ten runs with different random number seeds. These are compared to the probability of initiation as predicted by the *poe* method (with its batch statistics) as shown in Table II. Both algorithms were run with 100,000 initial families and the *poi* method had a divergent threshold of 1,000 progeny.

Table II. Uranium / Plutonium test results

Case	POI	σ	POE's poi	σ
U_195	0.095	0.001	0.096	0.001
U_225[a]	0.173	0.002	0.172	0.002
U_225[b]	0.238	0.001	0.240	0.002
U_285	0.291	0.001	0.292	0.001
U_315	0.336	0.001	0.337	0.002
Pu_120	0.093	0.001	0.092	0.002
Pu_140	0.174	0.002	0.175	0.001
Pu_160	0.245	0.002	0.249	0.003
Pu_180	0.302	0.001	0.307	0.006

[a] For density of 30.3 g/cc.
[b] For density of 36.6 g/cc

The case name in Table II corresponds to the density multiplied by the radius of the sphere. The material is represented by a U for uranium or Pu for plutonium. The *poe* results differ slightly from previous results [4] due to running a higher precision run with ten times the number of progeny. The probabilities for these test cases agree with each other to within the standard deviation for the *poe* and *poi* methods.

3.3. Time Step Testing

The *poi* and *poe* methods behave differently with the number of time steps. The *poi* method tends to build up simulation particles until it hits the divergence threshold. At this point, it starts to delete families of particles and processes the remaining families. So a typical run involves an initial phase in which each cycle gets slower as the simulation particle population builds up. This initial phase is followed by a speed-up as families are destroyed and there is decreasing work for each cycle.

The *poe* method attempts to keep the same amount of work for each cycle, so each cycle roughly runs as fast as the previous cycle. A disadvantage of this approach is that Russian Rouletting early in the problem may seed noise in the problem. Small time steps may also lead to increased noise.

In addition, if the total simulation time is an overestimate of the time it takes to reach an asymptotic value, the *poi* method will speed through these extraneous cycles, while the *poe* method will spend equal time on these cycles.

A simple test was created to investigate these issues. The test problem is a sphere of uranium-235 with a radius of 10 cm and a density of 18 g/cm^3. The source is a 1 MeV point source in the center of the sphere for 100,000 families. The *poi* threshold for divergence is set to 1,000 and the total simulation time is set to 1 μs. The number of cycles is varied along with the safety flag for the *poe* method. The results are presented in Table III.

Table III. Time step change results

	POI			POE with Safety On			POE with Safety Off		
# Cycles	Time	σ	FOM	Time	σ	FOM	Time	σ	FOM
100	129	0.0008	226	23	0.0028	103	11	0.0088	24
50	70	0.0012	182	12	0.0019	457	11	0.0069	44
25	81	0.0011	203	28	0.0015	319	13	0.0027	214
10	152	0.0010	134	Out of Memory			28	0.0011	567

The time is in seconds on 24 processors of 2.8 GHz Intel Xenon Hex machine with 2 gigabytes RAM per processor. The figure of merit is given by Eq. (2):

$$FOM = \frac{1}{(\sigma / poi)^2 (runtime)} \tag{2}$$

For the *poi* algorithm, the FOM roughly stays the same as the number of cycles is varied. For the 100 cycle case, the runtime is slowed by the end-of-cycle tallying, cleanup and parallel communications. At the 10 cycle case, the run time increases because there are only ten chances for large divergent families to be removed from the problem.

The *poe* algorithm with the safety on exhibits a slightly different behavior. For a large number of cycles, the run time is longer due to end-of-cycle computations and the statistical uncertainty is higher due to early Russian Rouletting seeding changes. As the number of cycles is decreased, only half of the population can be killed at most due to population control. So excess particles are kept, which improves the answer but results in extra simulation particles. For the 10 cycle case, the code crashed due to running out of memory from all the excess simulation particles.

With the safety off, the *poe* method is excessively aggressive in killing simulation particles early in the 100 cycle case. This results in a higher uncertainty with a shorter run time. However, at 25 cycles, the benefit of stronger population control starts to even out and by 10 cycles, the low uncertainty and run time result in a large FOM.

3. DISCUSSION

The *poi* and *poe* methods in mercury have both advantages and disadvantages. The *poi* method relies upon the user knowing the threshold number of progeny to determine a successful chain while the *poe* method requires no such knowledge. The *poi* method could be enhanced to detect if an inaccurate threshold value were chosen based on the estimate for the probability of initiation. A low probability of initiation requires a higher threshold.

The *poi* method has a population control method that allows for it to run an analog simulation whereas the *poe* method introduces population control early on, which could introduce statistical noise. The *poe* performance is correlated very strongly with the safety flag and the number of time steps.

The *poe* method has features that have not yet been implemented into the *poi* method such as batch statistics and a projected *poi* value. These features should be merged into the existing *poi* algorithm.

Since both *poi* and *poe* have a slightly different emphasis on solution, running both methods is one suggested way to calculate the probability of initiation. Differences in answers may mean that a longer time interval should be used, the progeny threshold should be raised for *poi*, or statistical problems could exist in *poe*. If statistical problems exist in *poe*, turning the safety off may allow for fewer time steps and eliminate excessive statistical noise due to population control.

4. FUTURE WORK

Some of the enhancements implemented into the *poe* method, such as batch statistics and a projected probability for particles still in flight, should be applied to the *poi* method. Both methods have memory constraints due to keeping a record of all families on each processor, and this currently limits the number of starting families to the few million range. This limit could be removed by batching several runs together in which each batch handles a portion of the starting families. In addition, it may be advantageous to look at a hybrid method of the *poi* and *poe* methods that could try to realize the benefits of both. Finally, an investigation is planned to see if modifying the cross sections may lead to a different *poe* method which would not have the same memory constraints.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The authors would like to thank Tom Booth and Malvin Kalos for their helpful guidance in these methods.

REFERENCES

1. G. I. Bell, “On the Stochastic Theory of Neutron Transport,” *Nucl. Sci. Eng.*, **21**, 390-401 (1965).
2. B. Mechitoua, ‘Monte Carlo Estimates of Nonextinction Probabilities,” *Trans. Am. Nucl. Soc.*, **82**, 134-135 (2000).
3. P. S. Brantley and M. S. McKinley (editors), “The Mercury Monte Carlo Web Site,” Lawrence Livermore National Laboratory, Web Document LLNL-WEB-500112, http://mercury.llnl.gov (2012).
4. G. M. Greenman, R. J. Procassini and C. J. Clouse, “A Monte Carlo Method for Calculating Initiation Probability,” *Proc. Joint Int. Topl. Mtg. Mathematics and Computations and Supercomputing in Nuclear Applications (M&C + SNA 2007)*, Monterey, CA, April 15-19, American Nuclear Society, CD-ROM (2007).
5. T. E. Booth, “Comments on Monte Carlo Probability of Initiation Estimates for Neutron Fission Chains,” *Nucl. Sci. & Eng.*, **166**, pp. 175-178 (2010).
6. T. E. Booth, “Monte Carlo Probability of Initiation Estimates in MCNP,” LA-UR 09-05874, Los Alamos National Laboratory (2009).
7. M. E. Rising, F. B. Brown and A. K. Prinja, “The Probability of Initiation in MCNP,” *Trans. Am. Nucl. Soc.*, **102**, pp. 258-260 (2010).
8. M. J. O’Brien, *et. al.*, “Domain Decomposition of a Constructive Solid Geometry Monte Carlo Transport Code,” *2009 Int. Conf. on Adv. in Mathematics, Computational Methods, and Reactor Physics (M&C 2009)*, Saratoga Springs, NY, May 3-7 (2009).